Optimal Simulation of Two-Dimensional Alternating Finite Automata by Three-Way Nondeterministic Turing Machines

AUTHOR(S):

ITO, Akira; INOUE, Katsushi; TAKANAMI, Itsuo

数理解析研究所講究録
871巻 1994年 15-23

Optimal Simulation of Two-Dimensional Alternating Finite Automata by Three-Way Nondeterministic Turing Machines

伊藤暁(Akira ITO), 井上克司(Katsushi INOUE), 高浪五男(Itsuo TAKANAMI)
山口大学工学部

ABSTRACT We show that n log n space is sufficient for three-way nondeterministic Turing machines (3NTs) to simulate two-dimensional alternating finite automata (AFs), where n is the number of columns of rectangular input tapes. It is already known that n log n space is necessary for 3NTs to simulate AFs. Thus, our algorithm is optimal in the sense of space complexity point of view.

1. Introduction

Recently, Jiang et al. [1] has shown interesting properties of two-dimensional alternating finite automata (AFs). For example, the class of sets accepted by AFs are not closed under complementation, and two-dimensional alternating finite automata with only universal states (UFs) are not equal to the complements of two-dimensional nondeterministic finite automata (NAs). These results contradicts our earlier expectation with usual sense.

In order to draw them out, the same paper reveals the fact that three-way nondeterministic Turing machines (3NTs) requires at least n log n space to simulate UFs, where n is the number of columns of a rectangular input tape.

This paper will show that n log n space is also sufficient for 3NTs to simulate AFs. Combined with Jiangs' revelation, we can say that n log n space is necessary and sufficient, thus optimal for 3NTs to simulate AFs and UFs.

2. Preliminaries

In contrast with ordinary one-dimensional Turing machines [2] which are given one-dimensional strings as inputs, a two-dimensional Turing machine [3] is a off-line Turing machines whose input tapes are rectangular in shape. It can move around on the input tape to the left, right, up, or downward, but never falls off the boundary symbols #'s surrounding over the four border edges of the input tape. Well-known concepts on the ordinary automata theory, such as nondeterminism, alternation, one-way movement of input head, space complexity, etc., naturally extended to the two-dimensional automata, except slight modifications: One-way movement of the input head becomes here three-way movement of left, right, and downward directions. Also, two-dimensional space complexity functions here have two variables "m" and "n", which represent the number of rows and columns of the input tapes, respectively.

A *configuration* of two-dimensional alternating Turing machine M on input tape x is a triple $(x,(i,j),(q,\alpha,k,))$, where x is a rectangular input tape for M, (i,j) is the position of the input head, q is a state of the finite control, α is the content of the working tape, and k is the position of the working tape head. A two-dimensional alternating Turing machine M is called "L(m,n) space-bounded" if for each m, n ($m,n \geq 1$) and for each input tape x with m rows and n columns, there exists an accepting computation tree such that each node labeled with configuration $(x,(i,j),(q,\alpha, k))$ satisfies $|\alpha| \leq L(m,n)$, when M accepts x. Also, M is called "L(m) space-bounded" if for each m ($m \geq 1$) and for each square input tape x with m rows (m columns), there exists an accepting computation tree such that each node labeled with configuration $(x,(i,j),(q,\alpha,k))$ satisfies $|\alpha| \leq L(m)$, when M accepts x.

A two-dimensional finite automaton can be regarded as a two-dimensional Turing machine whose space complexity function is constantly bounded, i.e., $L(m,n) \leq c$ for some constant c.

Let "AF", "UF", and "3NT(L(m,n))" denote a *two-dimensional alternating finite automaton*, a *two-dimensional alternating finite automaton with only universal states*, and a *three-way nondeterministic L(m,n) space-bounded two-dimensional Turing machine*, respectively [3-5].

For any family of two-dimensional automata $\mathfrak{F}$, the class of sets of rectangular input tapes accepted by $\mathfrak{F}$ is denoted by $\mathcal{L}[\mathfrak{F}]$ and the class of sets of "square" input tapes accepted by $\mathfrak{F}$ is denoted by $\mathcal{L}[\mathfrak{F}^s]$, that is, the superscript "s" indicates the restriction of their input tapes to square ones.

For convenience, we simply denotes a configuration of AF M by a triple $(x,q,(i,j))$, where x is the input tape for M, q is a state of the finite control of M, and (i,j) is an input head position. Let $I_M(x)$ be the *initial configuration* $(x,q_0,(1,1))$ of M on x, where q_0 is the initial state of M. The set of all possible configurations of M on x is denoted by "$C_M(x)$". Let c and c' be two configurations of M. If c can reach c' by a single step, we write $c \vdash_M c'$.

Normally, the acceptance of alternating machines is defined in connection with the existence of an accepting computation tree.[3] It can, however, be defined by the following deterministic procedure, too. This is essentially the same scheme given by Chandra et al.[6]

```
main program GENERAL:
begin
if I_M(x) ∈ SATURATE(C_M(x),∅) then accept else reject                ...(1)
end.

function SATURATE(var D,A: subsets of C_M(x)): a subset of D - A
begin
E:= {c | c is an accepting configuration in D∪A};
loop do
   Δ := ∅;
   for each c ∈ D - E do
     if (c is existential and ∃c'∈E[c ⊢_M c'])
        or (c is universal and ∀c'(c ⊢_M c')[c'∈E])
     then Δ := Δ + {c}[1]
   endfor;
   E:= E+ Δ ;
   if Δ = ∅ then return E - A;
endloop
end.
```

Let each element of the set SATURATE($C_M(x),\emptyset$) after Step (1) of the program GENERAL be called "*generalized accepting configurations*," or "*g.a. configurations*" in short. In terms of [5], a generalized accepting configuration is the configuration c such that there exists a c-accepting computation tree (c-accepting computation tree itself is the computation tree whose root is labeled with c and whose leaves are all labeled with accepting configurations).

The set A as an argument of the function SATURATE is assumed to be that of configurations which had been already judged as g.a. configurations. In the constraint that the elements of A are fixed, the function SATURATE produces additional g.a. configurations only from the target set D, excluding the outer set $C_M(x) - D$ beyond D. We also neglect the part $D \cap A (\subseteq A)$ of D from the output range since it is already known as g.a. configurations.

The outer loop is repeated until there is no more configuration in D to be judged as g.a. configuration. At each step of the inner for-loop, each configuration in D which has not yet been judged as a g.a. configuration is tested by means of conjunction or disjunction on the previous judgment of its immediate successors. Thus, the algorithm proceeds in the bottom-up manner from the leaves (= accepting configurations) up to the root (= initial configuration) of the computation tree of M on x.

It should be noted that the final set of generalized accepting configurations will never change even if we evaluate it in any order. In other words, the calculating order is restricted only with the partial order relation $\vdash_M$. Based on this property of g.a. configurations, we can take another strategy. For example, we partition the set $C_M(x)$ into two disjoint parts C1 and C2 earlier, then alternatively change one part by the other as the scope of the evaluation for the function SATURATE. It is clear that this strategy will finally produces the same set of g.a. configurations as

---

1. For two sets A and B, "A + B " denotes the set $A \cup B - A \cap B$.

the program GENERAL:

```
main program MODIFIED:
begin {assume C1∪C2= C_M(x) & C1∩C2= ∅}
A:= ∅;
repeat
    ⊿A1:= SATURATE(C1,A);
    A:= A+ ⊿A1;
    ⊿A2:= SATURATE(C2,A);
    A:= A+ ⊿A2
until ⊿A2= ∅;
if I_M(x)∈A then accept else reject
end.
```

3. Results

In that follows, we show that two-dimensional alternating finite automata can be simulated by three-way nondeterministic n log n space-bounded Turing machines, where n is the number of columns of the input tapes. Note that this space bound function does not depend on the number m of rows of given input tapes.

Main Theorem. $\mathcal{L}[AF] \subseteq \mathcal{L}[3NT(n \log n)]$

Proof. Assume that an input tape x with m rows and n columns is given to an AF M. Let Q be the set of states of the finite control of M.

First of all, it should be noted that, with such a principle that the bottom area of x is saturated earlier than the top area, we can further modify the program MODIFIED described in the preceding section into a "recursive program" as follows.

```
main program RECURSIVE:
global const C[0..m+1]: where C[i]= {(x,q,(i,j)) | q∈Q & 0≤j≤n+1};
global var A[-1..m+2]: where A[i]⊆C[i] (0≤i≤m+1) and A[-1]=A[m+2]=∅;
begin
A[i]:= ∅ for each i(0≤i≤m+1);
ROLLER(0);
if I_M(x)∈A[1] then accept else reject
end.

procedure ROLLER(var i:row index of x):
begin
if i= m+2 then return;
loop do
   ROLLER(i+1);
   ⊿Ai:= SATURATE(C[i],A[i-1]∪A[i]∪A[i+1]);
   if ⊿Ai= ∅ then return;
   A[i]:= A[i]+ ⊿Ai
endloop
end.
```

Of course, SATURATE is the function described in the preceding section. Note that two configurations c=(x,q,(i,j)) and c'=(x,q',(i',j')) satisfy the

relation $c \vdash_M c'$ only when $|i-i'| \leq 1$. Thus, as actual arguments of SATURATE, we have only to employ the subset of A corresponding to the current row A[i] and its two adjacent rows A[i-1]∪A[i+1], not whole the elements of A.

The program RECURSIVE proceeds as if it presses and flattens "the ground level" by perpendicular moves of a "roller" with horizontal axis. Figure 1 illustrates the behaviors of the program for a computation graph, which is derived from the pair of some AF and some x. The reader can see from Figure 1 that configurations of the bottom area is evaluated as early as possible than those of the top area.

Here, we construct a three-way nondeterministic Turing machine M' that simulates the running process of the deterministic program RECURSIVE, not directly simulating the four-way alternating automaton M itself. Since M' can visit each row only once from the top row down to the bottom row on a given input tape, it must guess the consequences of the entire calls for the subroutine ROLLER evoked on the current row, which are not necessarily consecutive events.

In the following program, the array AL is used as an argument of SATURATE, which corresponds to A[i-1]∪A[i]∪A[i+1] of the program RECURSIVE. $\Delta G_k[1]$ is the guessed elements that would be added to A[i+1] by the kth execution of ROLLER(i+1). $\Delta A_k[0]$ corresponds to the output ΔAi from the function SATURATE. The two variables $\Delta AL0$ and $\Delta G_k[0]$ are used to check the correctness of the guesses. The integer variable maxc represents the number of times of calls for the subroutine ROLLER on the preceding i-1st row. Below, K(Q,n) denotes the set $\{(q,j) \mid q \in Q \ \& \ 0 \leq j \leq n+1\}$ and q_0 is the initial state of M.

```
main program THREE-WAY_NONDETERMINISTIC:
var AL[-1..1]: where AL[r]⊆K(Q,n) (-1≤r≤1);
    ΔAL0: where ΔAL0⊆K(Q,n);
    ΔGk[0..1] (k≥1): where ΔGk[r]⊆K(Q,n) (0≤r≤1);
    ⊿Ak[-1..0] (k≥1): where ⊿Ak[r]⊆K(Q,n) (-1≤r≤0);
begin
maxc:=1;
for i=0 to m+1 do
   move to the ith row (when i=0, assume #'s on the 1st row);
   AL[r]:=∅ for each r(-1≤r≤1);
   k:=1;
   for ℓ=1 to maxc do
      ΔAL0:=∅;
      loop do
         if k≥2 and ⊿Ak-1[0]=∅ then ΔGk[1]:=∅                    ...(2)
         else if i=m+1 then ΔGk[1]:=∅                            ...(3)
            else guess ΔGk[1]⊆K(Q,n)-AL[1];
         AL[1]:=AL[1]+ΔGk[1];
         ⊿Ak[0]:=SATURATE({(x,q,(i,j)) | q∈Q & 0≤j≤n+1},AL);
         if ⊿Ak[0]=∅ then exit loop;
         AL[0]:=AL[0]+⊿Ak[0];
         ΔAL0:=ΔAL0+⊿Ak[0];
         k:=k+1
      endloop;
      if i≠0 and ΔAL0≠ΔGℓ[0] then reject;                        ...(4)
      if i≠0 then AL[-1]:=AL[-1]+⊿Aℓ[-1]
   endfor;
   maxc:=k;
   if i=1 and (q0,1)∉AL[0] then reject;
   for k=1 to maxc do
      ⊿Ak[-1]:=⊿Ak[0];
      ΔGk[0]:=ΔGk[1]
   endfor
endfor;
accept
end.
```

When the latest value of the output ⊿Ai from the function SATURATE of the former program RECURSIVE, is empty (except when the very first call of it on each row), our simulating program stops the honest trace of the further recursion, which will be evoked here, since there would be no change on the array variable A. In order to save the wasteful job of this kind, it simply sets ΔGk[1] to empty set as shown in Step (2).

On the *i*th ($0 \le i \le m$) row, the correctness of each guess $\Delta G_\ell[0]$, which had been guessed on the preceding i-1st row, is checked at Step (4) by the value of ⊿Ak[0], which has been probably influenced by the recent guess ΔGk[1] during the computation of the function SATURATE.

On the m+1st row, we can faithfully fix the value of ΔGk[1] to empty set for each k as shown in Step (3), since the recursive call for ROLLER(m+2) evoked here immediately returns without anything done. Therefore, the correctness of all the guesses described above is lastly confirmed when M' visits the bottom boundary row.

Table.1 illustrates the process when the key variables of the program are changing their contents in an accepting computation of M' on the computation

graph shown in Figure 3.

It is clear that the program above correctly simulates the program RECURSIVE, thus T(M')=T(M). We omit here the proof of the fact that the amount of space used by M' is bounded by O (n log n).□

Fact.(1) If L(m)= o (m log m), then $\mathcal{L}$[UF^s] $\not\subseteq$ $\mathcal{L}$[3NT^s(L(m))].

From this (and the obvious fact $\mathcal{L}$[UF]$\subseteq$ $\mathcal{L}$[AF]), we get the following.

Corollary. n log n space is necessary and sufficient for 3NTs to simulate AFs (UFs).

It is known [1] that n space is necessary and sufficient for 3NTs to simulate the complements of AFs (the necessity is straightforward).

References

[1] T.Jiang,O.H.Ibarra,and H.Wang, Some results concerning 2-D on-line tessellation acceptors and 2-D alternating finite automata, *Lect.Notes in Comp.Sci.* **520**, pp.221-230 (1991).

[2] J.E.Hopcroft and J.D.Ullman, *Introduction to Automata Theory, Languages, and Computation*, Addison Wesley (1979).

[3] K.Inoue,I.Takanami,and H.Taniguchi, Two-dimensional alternating Turing machines, *Theoret.Comput.Sci.* **27**, pp.61-83 (1983).

[4] A.Ito,K.Inoue,I.Takanami,and H.Taniguchi, Two-Dimensional Alternating Turing Machines with Only Universal States, *Inform. & Contr.* **55**, Nos1-3 (1982) 193-221.

[5] A.Ito, K.Inoue, and I.Takanami, Deterministic Two-dimensional On-Line Tessellation Acceptors are Equivalent to Two-Way Two-Dimensional Alternating Finite Automata through 180°-Rotation, *Theoret.Comput.Sci.* **66** pp.273-287 (1989).

[6] A.K.Chandra,D.C.Kozen,and L.Stockmeyer, Alternation, *J.Ass.Comp.Mach.* **28**, pp.114-133 (1981).

[7] A.Ito,K.Inoue,I.Takanami,and H.Taniguchi, Relationships of the Accepting Powers between Cellular Space with Bounded Number of State-Changes and Other Automata, *Trans.IECE Japan* **J68-D**, No.9, pp.1562-1570 (Sep.1985), in Japanese [translated to *Systems and Computers in Japan* **17**, No.7, pp.63-72 (1986)].

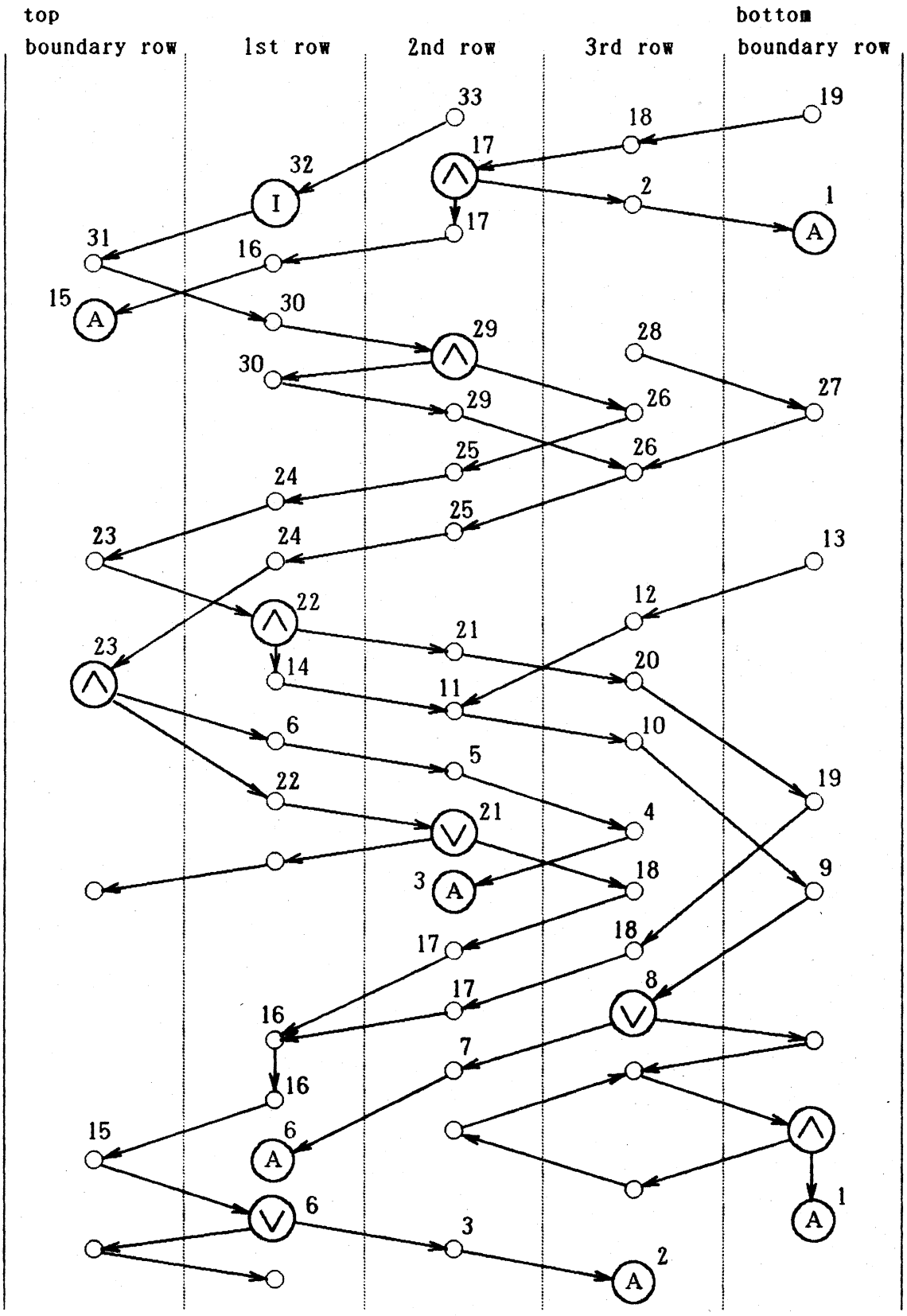

Fig.1 Behaviour of the program RECURSIVE on some computation graph. Script number denote the order of configurations when they are judged as generalized accepting computations.

Table.1 Running process of the program THREE-WAY_NONDETERMINISTIC lead to the acceptance.†

top boundary row		1st row		2nd row		3rd row		bottom boundary row	
$\Delta G_k[1]$	$\Delta A_k[0]$	$\Delta G_k[1]$	$\Delta A_k[0]$	$\Delta G_k[1]$	$\Delta A_k[0]$	$\Delta G_k[1]$	$\Delta A_k[0]$	$\Delta G_k[1]$	$\Delta A_k[0]$
								ϕ	1
								ϕ	ϕ
						1	2	[ϕ]	ϕ
						ϕ	ϕ		
				2	3	[ϕ]	4	[ϕ]	ϕ
						ϕ	ϕ		
				4	5	[ϕ]	ϕ		
				ϕ	ϕ				
						[ϕ]	8	[ϕ]	9
				[ϕ]	7			ϕ	ϕ
						9	10	[ϕ]	ϕ
		3, 5	6			ϕ	ϕ		
				8, 10	11	[ϕ]	12	[ϕ]	13
								ϕ	ϕ
						13	ϕ		
				12	ϕ				
		7, 11	14	[ϕ]	ϕ				
		ϕ	ϕ						
						[ϕ]	18	[ϕ]	19
				[ϕ]	17			ϕ	ϕ
						19	20	[ϕ]	ϕ
		[ϕ]	16			ϕ	ϕ		
6, 14	15			18, 20	21	[ϕ]	ϕ		
				ϕ	ϕ				
		17, 21	22	[ϕ]	ϕ				
		ϕ	ϕ						
						[ϕ]	26	[ϕ]	27
				[ϕ]	25			ϕ	ϕ
						27	28	[ϕ]	ϕ
		[ϕ]	24			ϕ	ϕ		
16, 22	23			26, 28	29	[ϕ]	ϕ		
				ϕ	ϕ				
		25, 29	30	[ϕ]	ϕ				
		ϕ	ϕ						
				[ϕ]	33	[ϕ]	ϕ		
		[ϕ]	32	ϕ	ϕ				
24, 30	31								
		33	ϕ						
32	ϕ								
maxc= 4		maxc= 11		maxc= 17		maxc= 19		maxc= 15	

† Here, each number denotes the configuratons which are told off in such a number in Fig.3. Note that other variables $\Delta G_g[0]$ and $\Delta A_g[-1]$ becomes $\Delta G_k[1]$ and $\Delta A_k[0]$ in the next row, respectively. The symbol "[ϕ]" denotes the empty set generated at Step (2) of the program.